Introduction
Module for ELEC 301 project

What is Matrix Completion

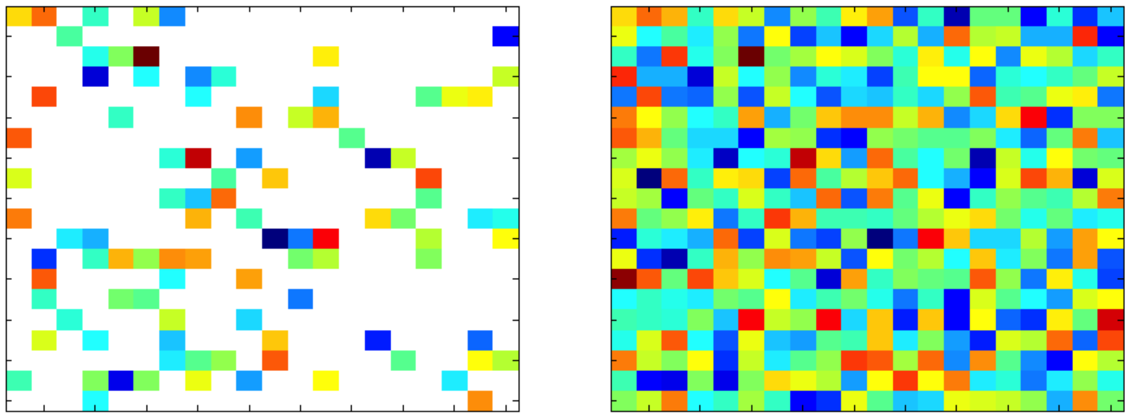
In 2006, Netflix issued a million dollar challenge to the world:

“Is there a computer algorithm that can accurately predict a user’s movie preferences?”

In the contest, a data matrix was given that contained ratings of thousands of movies from thousands of examinees, but it was only 2% completed. Contestants for this Netflix Challenge had to complete the matrix and provide the optimal algorithms for the task.

The Netflix Prize was won in 2009, but the ideas and algorithms generated to complete matrices remain vast and powerful in real world applications. Simply put, the Matrix Completion algorithm can be used for any areas that involve using a data matrix.

From a more scientific perspective, the 2008 paper, Exact Matrix Completion via Convex Optimization by Candes and Recht formalized a majorization minimization algorithm for matrix completion. Eric Chi’s 2014 article Getting to the Bottom of Matrix Completion and Nonnegative Least Squares with the MM Algorithm provides a more grounded framework for the problem and explains the mathematical concepts behind matrix completion.



A visual representation of matrix completion.

A visual representation of matrix completion.

When Does Matrix Completion WOrk

Given a sparse matrix with movies along one axis and users along another, the algorithm had to predict how those users would rate movies they have not seen. The solution, known as Matrix Completion, provided a good estimate of sparse data, provided it satisfied the following:

1. The matrix must be low rank
2. The unobserved indices in the matrix must be uniformly distributed

In terms of the Netflix Problem, the matrix was extremely sparse -- with millions of users and movies, less than 2% of the matrix was actually filled. The matrix also followed the above assumptions, specifically that there are a few “types” of people who watch Netflix (an action movie lover, a rom-com fanatic, etc.), making it low rank, and that each user’s reviews are spread uniformly throughout the matrix.

Characterizing the Problem

Often, in the real world, these idealities are not upheld. It is very rare to find a matrix that is both perfectly uniform and low rank. In order to better understand matrix completions’ application to the real world, our project aimed to stretch the second requirement and better characterize the algorithm’s limits.

Specifically, we decided to focus on the requirement that the unobserved indices in the matrix must be uniformly distributed. How uniform do the unobserved entries need to be? At what point does matrix completion “stop working”?

Even more importantly, what does a plot of the error look like as a function of uniformity? We know that non-uniform data will result in a predicted matrix that is very dissimilar to the actual matrix, and we know that uniform data will result in a predicted matrix that very similar to the actual data, but what happens in between? Does a small amount of non-uniformity result in an unusable matrix, or can matrix completion continue to work under less than ideal conditions?

Real World Implications

While it is important to characterize algorithms to have a better theoretical understanding of how and when they work, this research has very salient real world applications as well.

Imagine an old picture with non-uniform noise distributed throughout it -- maybe one area of the photo is particularly noisy. If matrix completion can work in the conditions described previously, it would be able to reconstruct those images.

Even more importantly, matrix completion is used to predict cancer survival rates, among other medical applications. There is no guarantee that this data is uniform, but maybe matrix completion can still be trusted in these situations despite this limitation.

Implementation of Matrix Completion
Summarizes the implementation of matrix completion using Chis paper

Objective of Matrix Completion

$$\text{minimize } \frac{1}{2}\|P_{\Omega^c}(\mathbf{X}) - P_{\Omega^c}(\mathbf{Z})\|_F^2 + \lambda\|\mathbf{Z}\|_*,$$

Equation 1 : Objective of the matrix completion

The objective of matrix completion is to minimize Equation 1 stated above. X is the matrix of actual data, and Z is our prediction model. In real world situations, such as the Netflix Problem, the actual data X is only partially filled. Thus, the term P**Ω**c(X) represents the observed indices of the data matrix X, and the following term represents the observed indices of our model matrix Z. The first half of Equation 1, the Frobenius norm of the differences between the observed X and the model Z, would therefore signify how closely the model resembles the actual data.

The second half of Equation 1, the nuclear norm of the model matrix Z, is called the regularization term, which is used here to represent the rank of the model, which is an appropriate indicator of the simplicity of the model. Simple matrices would have smaller quantities and magnitudes of singular values.

However, as the equation shows, there exists a “tradeoff between the [simplicity] (rank) of the model and how well the model matches the data.” If the model is too simple, it is often not accurate. If the model is perfectly accurate, it is often not simple.

Overview of Majorization Minimization Algorithm

The first half of Equation 1 still provides a challenge because it only uses terms that are observed. The question that can be asked at this point would be how can we convert the projection matrix, P**Ω**c(X), into a fully completed matrix and still provide the same result for the model Z?

Here we introduce the majorization-minimization (MM) algorithm. The following explanation will give an overview of the algorithm applied to f(x).

The first step of the algorithm is majorizing the function f(x). Majorizing means finding a good surrogate of the actual function f(x) anchored at a point xn. This means that the surrogate function g(x) must have the same value with f(x) at xn. g(x) must always be greater than f(x) at any point of x. In other words, g(x) must dominate f(x).

After we find the majorization function g(x), the second step of the algorithm is minimization, which means to find the lowest value of g(x). x at the lowest value of g(x) would be our next anchor for the majorization of the next iteration of MM algorithm.

In the next part, we will look at how we implement the MM algorithm into the matrix completion problem.

Majorization for Matrix Completion Problem

$$\begin{aligned}\frac{1}{2}\|P_{\Omega^c}(\mathbf{X}) - P_{\Omega^c}(\mathbf{Z})\|_{\mathrm{F}}^2 &\le \frac{1}{2}\sum_{(i,j)\in\Omega^c}(x_{ij} - z_{ij})^2 + \frac{1}{2}\sum_{(i,j)\in\Omega}(z_{ij} - z_{ij}^{(n)})^2 \\ &= \frac{1}{2}\sum_{(i,j)}(y_{ij} - z_{ij})^2 \\ &= \frac{1}{2}\|\mathbf{Y} - \mathbf{Z}\|_{\mathrm{F}}^2,\end{aligned}$$

where

$$y_{ij} = \begin{cases} x_{ij} & \text{for } (i,j) \notin \Omega, \\ z_{ij}^{(n)} & \text{for } (i,j) \in \Omega. \end{cases}$$

Using the quadratic majorization of the matrix, we can therefore simplify the original problem with the surrogate matrix Y written below:

$$\text{minimize } \frac{1}{2}\|\mathbf{Y} - \mathbf{Z}\|_{\mathrm{F}}^2 + \lambda\|\mathbf{Z}\|_*.$$

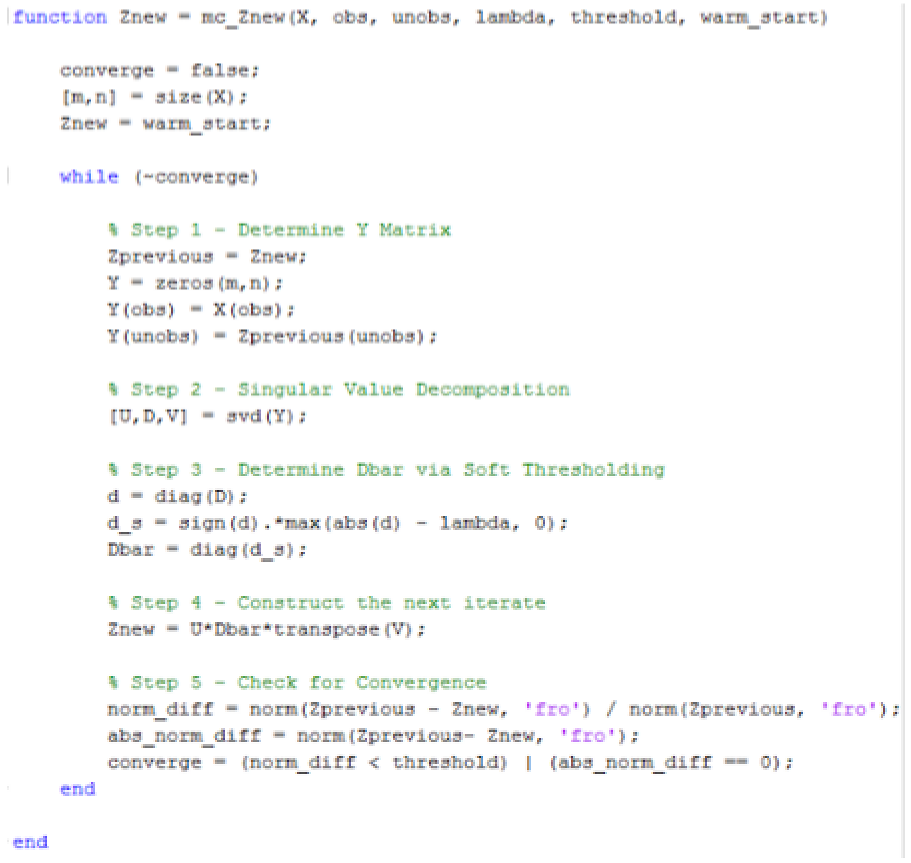
Minimization for Matrix Completion Problem (Soft Threshold Operator and 4 steps)

1. $y_{ij}^{(n+1)} = \begin{cases} x_{ij} & \text{for } (i,j) \notin \Omega, \text{ and} \\ z_{ij}^{(n)} & \text{for } (i,j) \in \Omega. \end{cases}$
2. $(\mathbf{U}, \mathbf{D}, \mathbf{V}) = \mathrm{SVD}[\mathbf{Y}^{(n+1)}]$
3. $\bar{\mathbf{D}} = S(\mathbf{D}, \lambda)$
4. $\mathbf{Z}^{(n+1)} = \mathbf{U}\bar{\mathbf{D}}\mathbf{V}^t$

Given the majorization matrix Y, we now implement the minimization using the 4 steps shown above, which include building a Y matrix, singular value decomposition of Y, soft thresholding of the ranks, and building the next model matrix Z. Soft thresholding, is essentially the solution to the minimization of the majorization of Equation 1. The soft thresholding process is shown below:

$$\begin{aligned} S(s,\lambda) &= \arg\min \frac{1}{2}(s-t)^2 + \lambda|t| \\ &= \begin{cases} s-\lambda & \text{if } s \geq \lambda, \\ s+\lambda & \text{if } s \leq -\lambda, \text{ and} \\ 0 & |s| < \lambda. \end{cases} \end{aligned}$$

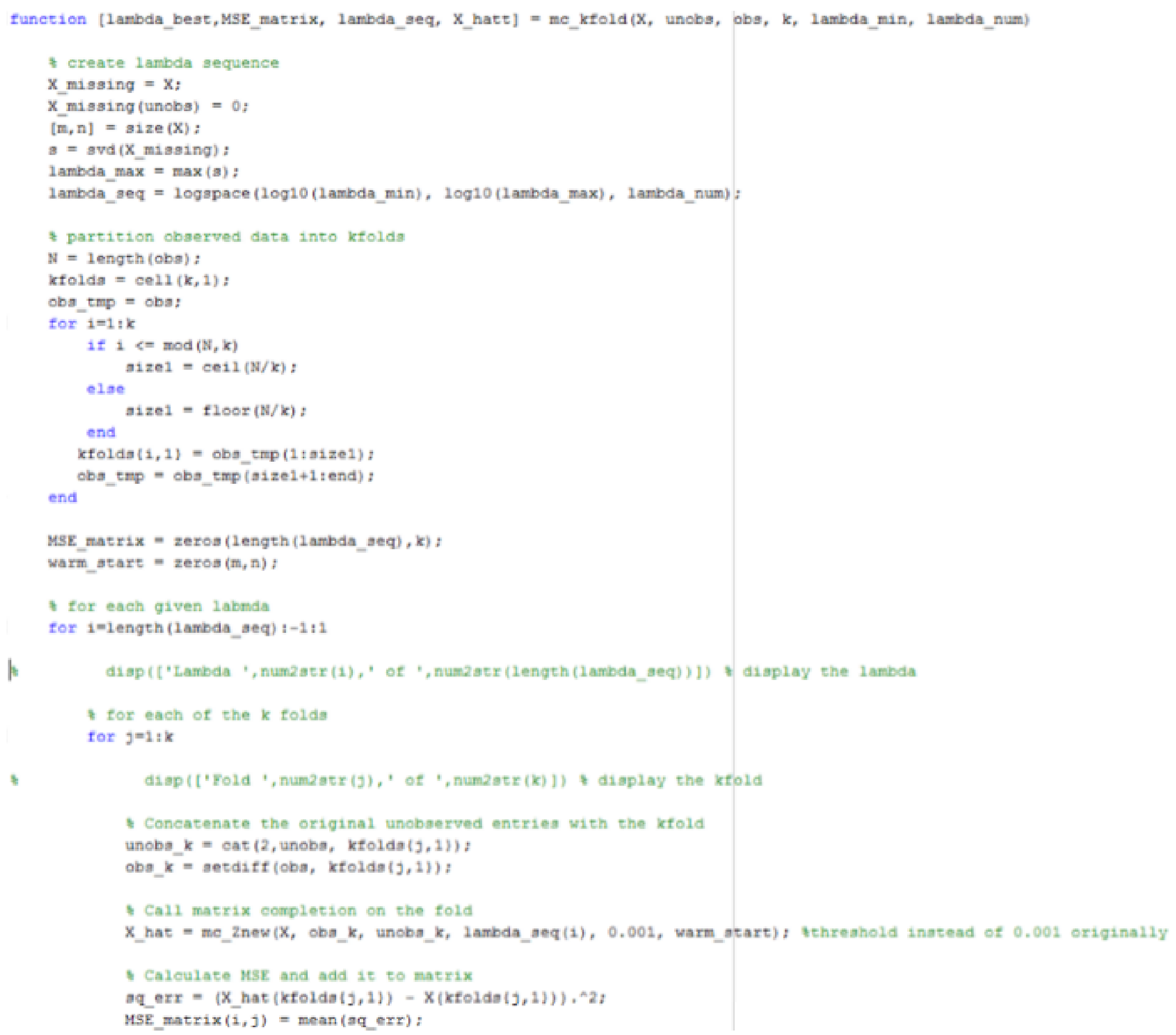
The detailed proof of achieving minimization through the soft thresholding process can be found on the articles referenced at the end of the report. (“Getting to the Bottom of Matrix Completion”) The relevant code we wrote is shown below:



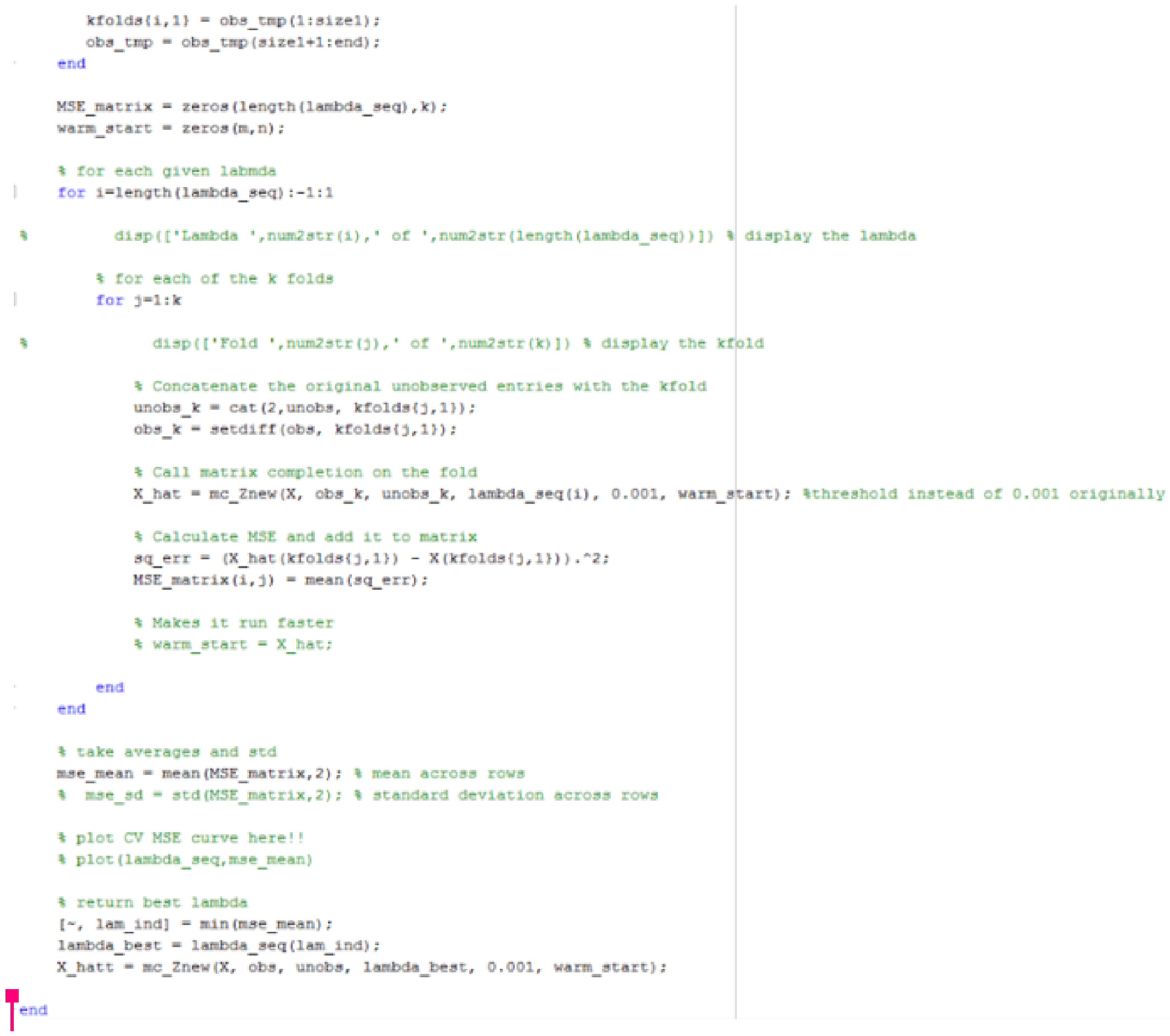
```
function Znew = mc_Znew(X, obs, unobs, lambda, threshold, warm_start)

    converge = false;
    [m,n] = size(X);
    Znew = warm_start;

    while (~converge)

        % Step 1 - Determine Y Matrix
        Zprevious = Znew;
        Y = zeros(m,n);
        Y(obs)  = X(obs);
        Y(unobs)  = Zprevious(unobs);

        % Step 2 - Singular Value Decomposition
        [U,D,V] = svd(Y);

        % Step 3 - Determine Dbar via Soft Thresholding
        d = diag(D);
        d_s = sign(d).*max(abs(d) - lambda, 0);
        Dbar = diag(d_s);

        % Step 4 - Construct the next iterate
        Znew = U*Dbar*transpose(V);

        % Step 5 - Check for Convergence
        norm_diff = norm(Zprevious - Znew, 'fro') / norm(Zprevious, 'fro');
        abs_norm_diff = norm(Zprevious- Znew, 'fro');
        converge = (norm_diff < threshold) | (abs_norm_diff == 0);
    end

end
```

K-fold Cross Validation

We used the 10-fold cross validation process to find the optimal regularization term lambda λ. The process includes randomly dividing the observed indices into 10 folds. Then, we remove each fold and use the rest of the indices to find the model Z. For each fold, therefore, we find the mean squared difference of the observed indices between data X and model Z. We then average the results of each 10 fold to get a comprehensive idea of how successful the lambda was. We iterate this process for thousands of lambdas to find the optimal value. The matlab code we wrote is shown below:



```
function [lambda_best,MSE_matrix, lambda_seq, X_hatt] = mc_kfold(X, unobs, obs, k, lambda_min, lambda_num)

    % create lambda sequence
    X_missing = X;
    X_missing(unobs) = 0;
    [m,n] = size(X);
    s = svd(X_missing);
    lambda_max = max(s);
    lambda_seq = logspace(log10(lambda_min), log10(lambda_max), lambda_num);

    % partition observed data into kfolds
    N = length(obs);
    kfolds = cell(k,1);
    obs_tmp = obs;
    for i=1:k
        if i <= mod(N,k)
            size1 = ceil(N/k);
        else
            size1 = floor(N/k);
        end
       kfolds{i,1} = obs_tmp(1:size1);
       obs_tmp = obs_tmp(size1+1:end);
    end

    MSE_matrix = zeros(length(lambda_seq),k);
    warm_start = zeros(m,n);

    % for each given labmda
    for i=length(lambda_seq):-1:1

%         disp(['Lambda ',num2str(i),' of ',num2str(length(lambda_seq))]) % display the lambda

        % for each of the k folds
        for j=1:k

%             disp(['Fold ',num2str(j),' of ',num2str(k)]) % display the kfold

            % Concatenate the original unobserved entries with the kfold
            unobs_k = cat(2,unobs, kfolds{j,1});
            obs_k = setdiff(obs, kfolds{j,1});

            % Call matrix completion on the fold
            X_hat = mc_Znew(X, obs_k, unobs_k, lambda_seq(i), 0.001, warm_start); %threshold instead of 0.001 originally

            % Calculate MSE and add it to matrix
            sq_err = (X_hat(kfolds{j,1}) - X(kfolds{j,1})).^2;
            MSE_matrix(i,j) = mean(sq_err);
```



```
       kfolds{i,1} = obs_tmp(1:size1);
       obs_tmp = obs_tmp(size1+1:end);
    end

    MSE_matrix = zeros(length(lambda_seq),k);
    warm_start = zeros(m,n);

    % for each given labmda
    for i=length(lambda_seq):-1:1

%         disp(['Lambda ',num2str(i),' of ',num2str(length(lambda_seq))]) % display the lambda

        % for each of the k folds
        for j=1:k

%             disp(['Fold ',num2str(j),' of ',num2str(k)]) % display the kfold

            % Concatenate the original unobserved entries with the kfold
            unobs_k = cat(2,unobs, kfolds{j,1});
            obs_k = setdiff(obs, kfolds{j,1});

            % Call matrix completion on the fold
            X_hat = mc_Znew(X, obs_k, unobs_k, lambda_seq(i), 0.001, warm_start); %threshold instead of 0.001 originally

            % Calculate MSE and add it to matrix
            sq_err = (X_hat(kfolds{j,1}) - X(kfolds{j,1})).^2;
            MSE_matrix(i,j) = mean(sq_err);

            % Makes it run faster
            % warm_start = X_hat;

        end
    end

    % take averages and std
    mse_mean = mean(MSE_matrix,2); % mean across rows
    %  mse_sd = std(MSE_matrix,2); % standard deviation across rows

    % plot CV MSE curve here!!
    % plot(lambda_seq,mse_mean)

    % return best lambda
    [~, lam_ind] = min(mse_mean);
    lambda_best = lambda_seq(lam_ind);
    X_hatt = mc_Znew(X, obs, unobs, lambda_best, 0.001, warm_start);

end
```

Breaking Matrix Completion
Summary of how we stretched matrix completion to discover its limits.

The Stress Test

The main focus of our project was not to simply implement matrix completion -- that has been done many times in the past and is proven to work. What we wanted to do was put the algorithm through a “stress test” to understand its limits.

To do this, we devised the following plan:

1. Construct a low rank matrix
2. Create a model of uniformity
3. Vary the uniformity on unobserved indices in a matrix
4. Complete the matrix
5. Measure mean squared error between the original matrix and the completed matrix
6. Plot mean squared error as a function of uniformity

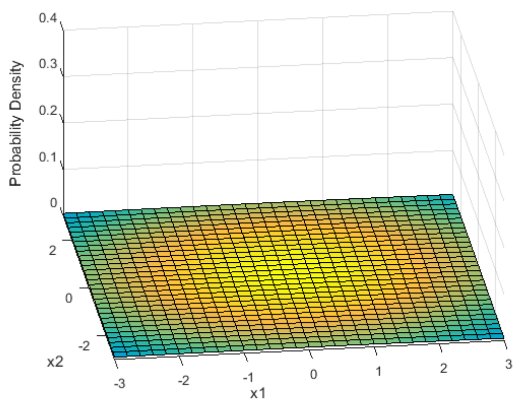
Constructing a Low Rank Matrix

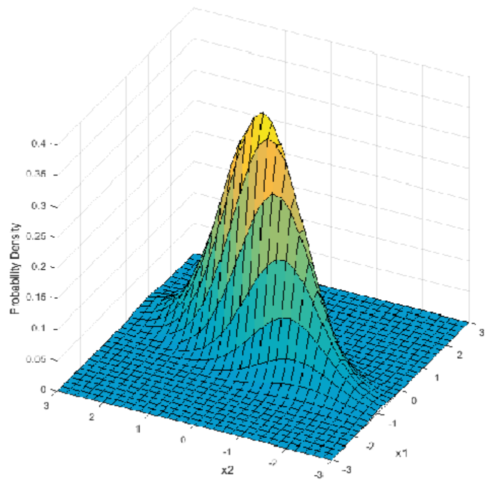
In order to construct a low rank matrix, remove indices, and compare the results of matrix completion with the actual matrix, we first created two separate matrices of size m x r and r x n with random entries between 0 and 1, where r in the desired rank. We then multiplied those two matrices together, constructing one matrix of size m x n and rank r.

Modeling Uniformity

Our second challenge was to generate a model of uniformity that provided a simple way to change how uniform the unobserved indices were. We decided to use a jointly Gaussian distribution, modifying the covariance to control uniformity. The idea is simple: low covariance results in a high concentration of indices removed from the center of the matrix, while high covariance results in an almost perfectly uniform distribution.

In the figure below, the matrix on the left shows a matrix with a very uniform distribution of unobserved entries, characterized by a jointly gaussian distribution with high covariance. The matrix on the right demonstrates a data set with a high degree of nonuniformity. The probability peaks at the center of the matrix, resulting in unobserved indices which are highly concentrated in the middle of the matrix.





Plotting MSE and Uniformity

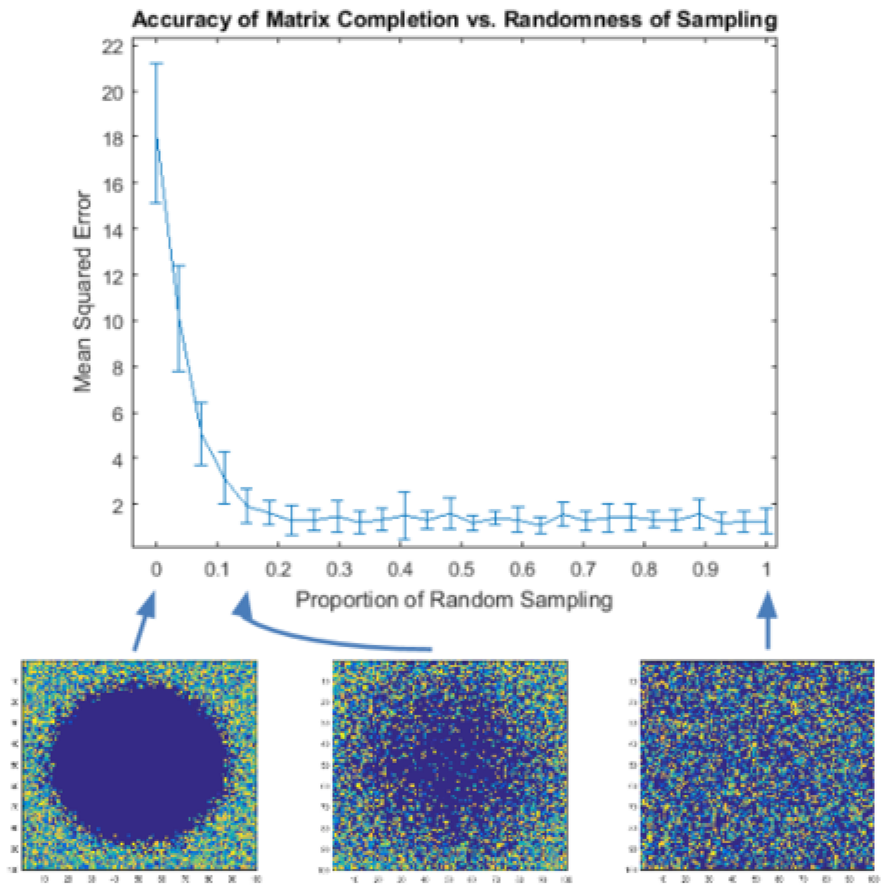
After completing the matrix, we calculated the mean squared error between the original matrix and the prediction provided by matrix completion and stored it in an array, along with the corresponding uniformity percentage. We calculated the uniformity percentage by normalizing each covariance by the largest covariance used, providing a scale between 0 and 100%.

Results
This is where we talk about the results of our project

Expected results

The first observation we made was the obvious -- the mean squared error for matrices that had unobserved indices which were completely non-uniform was very high, and had a large standard deviation, whereas the perfectly uniform matrices had a low mean squared error with a small standard deviation.

These results confirmed that our implementation of the algorithm and the code for the other parts of our project worked correctly.



Unexpected Results

The more surprising result of our tests, showed that matrix completion can tolerate a high level of non-uniformity. Our graph showed that at, according to our model, at about 15% uniformity, matrix completion returns results that are comparable to 100% uniformity.

Even the visual depictions of this kind of uniformity are interesting to ponder. At 15%, there is still a clear accumulation of unobserved indices at the center of the matrix, yet the algorithm works just as well.

The implications for this are tremendous, and the results show that uniformity isn’t a huge concern when applying matrix completion to real world problems. Unless the matrix is completely non-uniform, matrix completion will provide valid results.

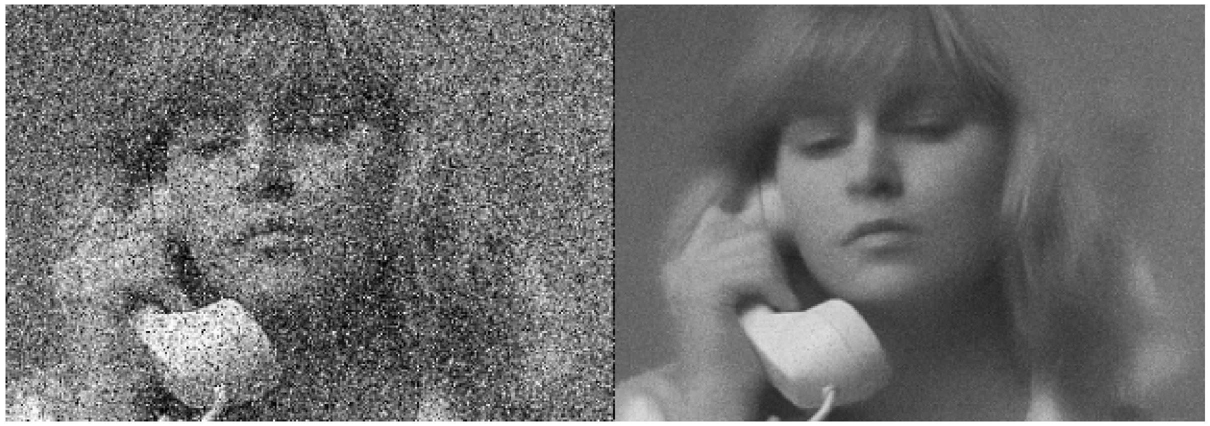
Future Work
This section details our plans for expanding on the project and its results

Generalizing Non Uniformity

As stated earlier, we wanted our measure of nonuniformity to be simple and easy to control. For the purposes of this project, removing entries based on a jointly Gaussian Distribution met our requirements. However, this measure of nonuniformity is both unrealistic and overly simplistic for real world data sets. With this in mind, future work would involve finding ways to generalize our notion of nonuniformity so that our results are more accurate with regards to real world data sets.

Testing with Real World Data

One of the most obvious applications of matrix completion is image reconstruction. The figure below shows how matrix completion can be used to reconstruct a noisy image. The original image had a uniform distribution of removed indices, so matrix completion was able to successfully recreate the image.



Our results suggest that the algorithm would similarly be able to complete the image if the removed indices had a certain degree of non uniformity. Further testing would involve looking for any kinds of exceptions to our results. Due to the visual nature of these matrices, it would be very easy to see when matrix completion succeeds or fails. We could then look for patterns to generalize the issues the algorithm takes with varying degrees of non uniformity in the missing indices.

Essentially, this is expanding the whole idea of our stress test to be more cognizant of real world issues of nonuniformity in data sets.

Alternative Approaches

This whole project has been guided by the principal that the existing solution for matrix completion is as good as it will get. At our poster presentation, one of the judge’s proposed an interesting idea. Could we (or others with better qualifications) improve the solution? Is there a way to fix the algorithm so that it works for all levels of uniformity?

We don't know the answer to this question and probably don't have the analytical skills to begin looking for one. However, a simple but potentially effective solution quickly comes to mind. We could develop a preprocessing algorithm that makes the data uniform before running matrix completion. Such an algorithm would need to be fast and generalized to all sorts of data sets to truly be a viable solution.

References

1 Tierney, Stephen. "An Introduction to Matrix Completion." SJTRNY. N.p., 4 Apr. 2014. Web. 09 Dec. 2015.

2 Chi, Jocelyn T., and Eric C. Chi. "Getting to the Bottom of Matrix Completion and Nonnegative Least Squares with the MM Algorithm." Statistics Views. N.p., 31 Mar. 2014. Web. 09 Dec. 2015.

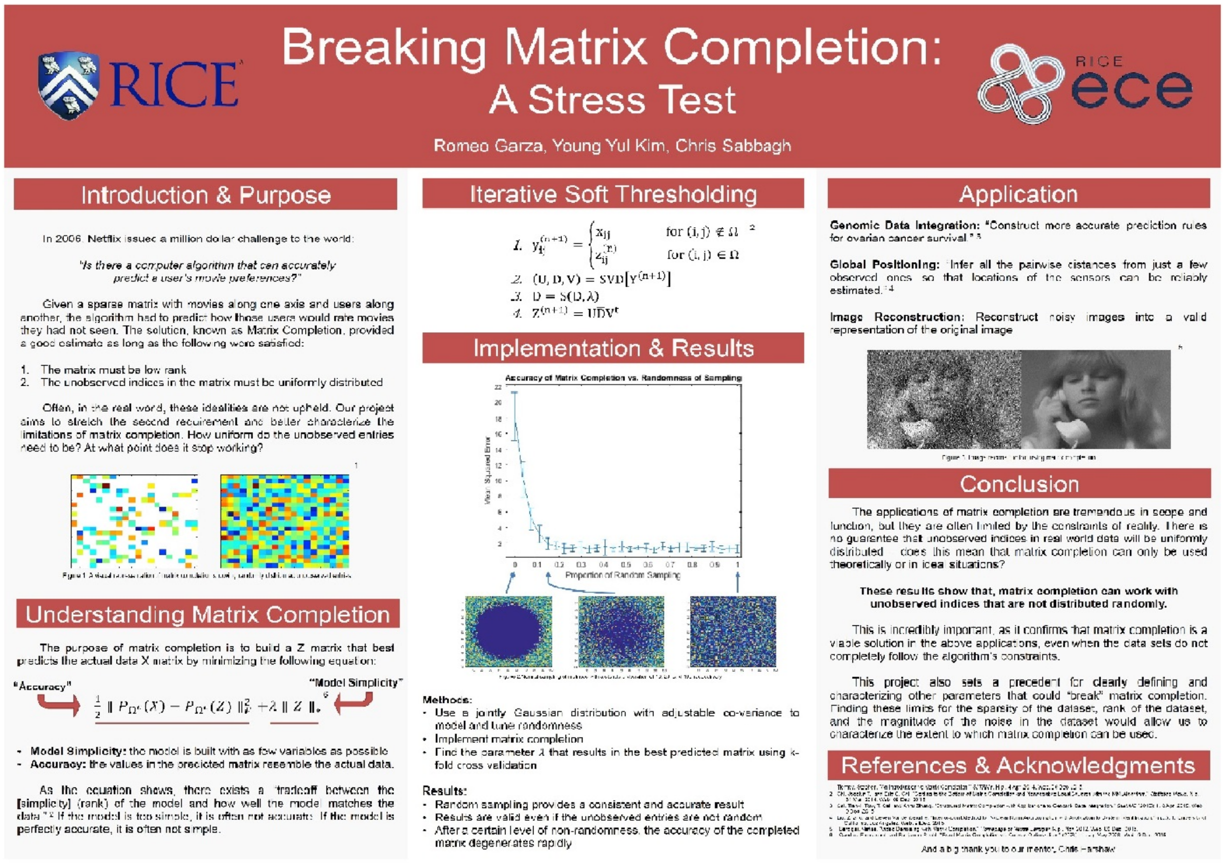
3 Cai, Tianxi, Tony T. Cai, and Anru Zhang. "Structured Matrix Completion with Applications to Genomic Data Integration." Stat.ME (2015): 1. 8 Apr. 2015. Web. 9 Dec. 2015.

4 Liu, Zhang, and Lieven Vandenberghe. "Interior-point Method for Nuclear Norm Approximation with Application to System Identification." (n.d.): 1. University of California, Los Angeles. Web. 9 Dec. 2015.

5 Barzigar, Nafise. "Video Denoising with Matrix Completion." Homepage of Nafise Barzigar. N.p., Nov. 2012. Web. 09 Dec. 2015.

6 Candes, Emmanuel, and Benjamin Recht. "Exact Matrix Completion via Convex Optimization." (2008): n. pag. May 2008. Web. 9 Dec. 2015.

Poster



Breaking Matrix Completion: A Stress Test

Romeo Garza, Young Yul Kim, Chris Sabbagh

Introduction & Purpose

In 2006, Netflix issued a million dollar challenge to the world:

"Is there a computer algorithm that can accurately predict a user's movie preferences?"

Given a sparse matrix with movies along one axis and users along another, the algorithm had to predict how those users would rate movies they had not seen. The solution, known as Matrix Completion, provided a good estimate as long as the following were satisfied:

1. The matrix must be low rank
2. The unobserved indices in the matrix must be uniformly distributed

Often, in the real word, these idealities are not upheld. Our project aims to stretch the second requirement and better characterize the limitations of matrix completion. How uniform do the unobserved entries need to be? At what point does it stop working?

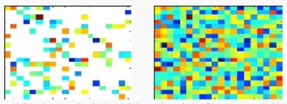

[1]

Figure 1: A visual representation of matrix completion showing randomly distributed observed entries

Understanding Matrix Completion

The purpose of matrix completion is to build a Z matrix that best predicts the actual data X matrix by minimizing the following equation:

"Accuracy" "Model Simplicity"

$$\underbrace{\frac{1}{2}\| P_{\Omega^c}(X) - P_{\Omega^c}(Z) \|_F^2}_{} + \lambda \underbrace{\| Z \|_*}_{}$$ [6]

- **Model Simplicity:** the model is built with as few variables as possible
- **Accuracy:** the values in the predicted matrix resemble the actual data.

As the equation shows, there exists a "tradeoff between the [simplicity] (rank) of the model and how well the model matches the data."[4] If the model is too simple, it is often not accurate. If the model is perfectly accurate, it is often not simple.

Iterative Soft Thresholding

$$1.\ \ y_{ij}^{(n+1)} = \begin{cases} x_{ij} & \text{for } (i,j) \notin \Omega \\ z_{ij}^{(n)} & \text{for } (i,j) \in \Omega \end{cases}$$ [2]

$$2.\ \ (U, D, V) = \mathrm{SVD}\left[Y^{(n+1)}\right]$$

$$3.\ \ \bar{D} = S(D, \lambda)$$

$$4.\ \ Z^{(n+1)} = U\bar{D}V^t$$

Implementation & Results

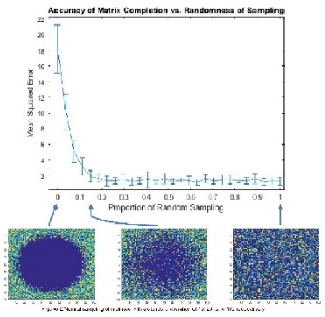


Figure 2: [illegible]

Methods:
- Use a jointly Gaussian distribution with adjustable co-variance to model and tune randomness
- Implement matrix completion
- Find the parameter λ that results in the best predicted matrix using k-fold cross validation

Results:
- Random sampling provides a consistent and accurate result
- Results are valid even if the unobserved entries are not random
- After a certain level of non-randomness, the accuracy of the completed matrix degenerates rapidly

Application

Genomic Data Integration: "Construct more accurate prediction rules for ovarian cancer survival."[3]

Global Positioning: "Infer all the pairwise distances from just a few observed ones so that locations of the sensors can be reliably estimated."[4]

Image Reconstruction: Reconstruct noisy images into a valid representation of the original image

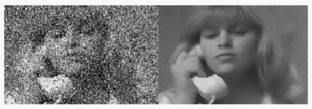

[5]

Figure 3: Image reconstruction using matrix completion

Conclusion

The applications of matrix completion are tremendous in scope and function, but they are often limited by the constraints of reality. There is no guarentee that unobserved indices in real world data will be uniformly distributed – does this mean that matrix completion can only be used theoretically or in ideal situations?

These results show that, matrix completion can work with unobserved indices that are not distributed randomly.

This is incredibly important, as it confirms that matrix completion is a viable solution in the above applications, even when the data sets do not completely follow the algorithm's constraints.

This project also sets a precedent for clearly defining and characterizing other parameters that could "break" matrix completion. Finding these limits for the sparsity of the dataset, rank of the dataset, and the magnitude of the noise in the dataset would allow us to characterize the extent to which matrix completion can be used.

References & Acknowledgments

1. Becker, Stephen. "An Introduction to Matrix Completion." [illegible] 2014. Web. [illegible] Dec. 2015.
2. Chi, Jocelyn T., and Eric C. Chi. "[illegible] Matrix Completion and [illegible]." [illegible] 31 Mar. 2014. Web. 06 Dec. 2015.
3. Cai, Tianxi, Tony T. Cai, and Anru Zhang. "Structured Matrix Completion with Applications to Genomic Data Integration." [illegible] 6 Apr. 2015. Web. 3 Dec. 2015.
4. [illegible] University of California, Los Angeles. Web. 3 Dec. 2015.
5. [illegible] "Robust [illegible] with Matrix Completion." [illegible] Nov. 2012. Web. 06 Dec. 2015.
6. [illegible] "Exact Matrix Completion via Convex Optimization." [illegible] May 2008. Web. [illegible] 2015.

And a big thank you to our mentor, Chris Harshaw